pomponio_alessandro_2020-01-07

February 8, 2021

1 Alessandro Pomponio

1.1 Matricola 0000920265

1. a pairplot of the data (see Seaborn pairplot) and a comment on remarkable situations, if any (5pt)
2. a classification model using a method of your choice with the schema "train-validationtest" exploring an appropriate range of parameter values (5pt)
3. the optimal parameter(s) (5pt)
4. a scatter plot of the test set using a pair of attributes of your choice with the class as colour (5pt)
5. … and the good/bad prediction as the point style (5pt)

[1]:
```
# Imports
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.metrics import accuracy_score

# Variables
filename = 'exam_2020_01_07_data.csv'
separator = ','
target = 'Class'
random_state = 42

# Directives
%matplotlib inline
np.random.seed(random_state)
```

[2]:
```
# Load the file and check that everything is in good shape
df = pd.read_csv(filename, sep = separator)
df.head()
```

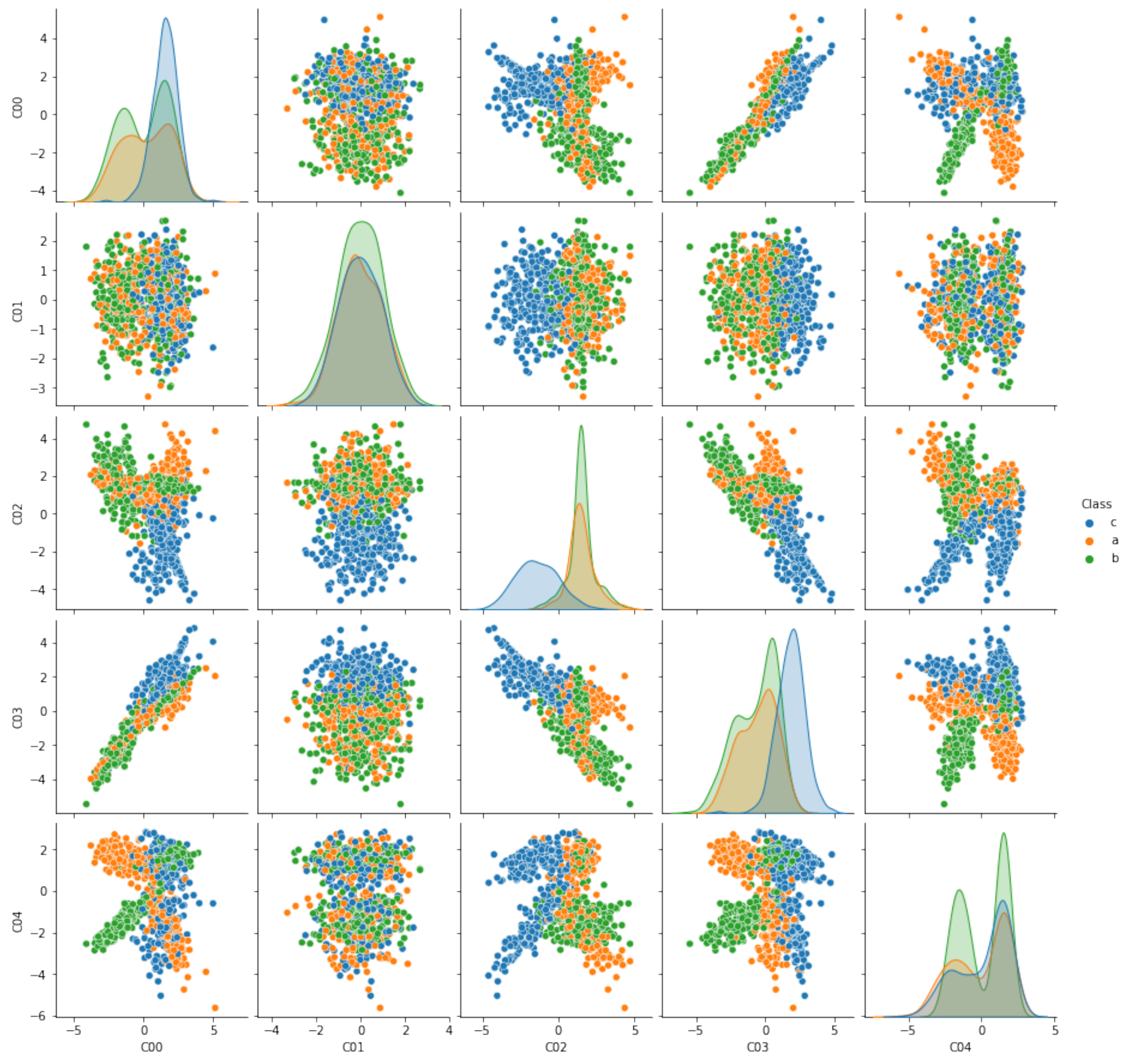
[2]:
```
        C00       C01       C02       C03       C04 Class
0  1.855416  0.466367 -0.176765  1.546514  0.149219     c
```

```
1 -0.107873 -0.136792  1.551591 -0.813810  1.357674     a
2  2.712560 -0.495846  1.397077  1.483562  1.656526     b
3 -2.166084 -0.582271  0.353011 -1.864210 -2.267033     b
4  2.848831 -0.507369  1.661752  1.466627  1.938519     b
```

1.1.1 1. A pairplot of the data (see Seaborn pairplot) and a comment on remarkable situations, if any (5pt)

[3]:
```python
# Show pairplots, use our target as hue to highlight any possible pattern
sns.pairplot(df, hue = target)
```

[3]:
```
<seaborn.axisgrid.PairGrid at 0x2140fea7070>
```



The pairplots don't seem to show any particular pattern in the data.

1.1.2 2. A classification model using a method of your choice with the schema "train-validation-test" exploring an appropriate range of parameter values (5pt)

For this classification assignment, we will use a Decision Tree.

We start by dividing the features matrix and the target column, as follows:

[4]:
```
# Divide our data between feature matrix and class labels
X = df.drop(target, axis = 1)
y = df[target]
```

We now divide the data in training and test set by means of the `train_test_split` function

[5]:
```
# We will use a 2/3 ratio between the training and test data
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, train_size = 2/3,␣
 ↪random_state = random_state)
print(f"We have {Xtrain.shape[0]} items in our training set")
print(f"We have {Xtest.shape[0]} items in our test set")
```

```
We have 666 items in our training set
We have 334 items in our test set
```

We now instantiate a Decision Tree and fit it on the training data.

We then use it to predict the training values, using the `accuracy_score` function to see the accuracy on the training set

[6]:
```
# Estimator creation and fitting
estimator = tree.DecisionTreeClassifier(criterion="entropy", random_state =␣
 ↪random_state)
estimator.fit(Xtrain, ytrain)

# Prediction and test set accuracy results
train_set_prediction = estimator.predict(Xtrain)
train_set_accuracy = accuracy_score(ytrain, train_set_prediction) * 100
print(f"The accuracy on the training set was {train_set_accuracy:.2f}%")
```

```
The accuracy on the training set was 100.00%
```

To have a more meaningful result, we will try it on the test set as well, to obtain a "baseline" value for the performance of our classifier

[7]:
```
test_set_prediction = estimator.predict(Xtest)
test_set_accuracy = accuracy_score(ytest, test_set_prediction) * 100
print(f"The accuracy on the test set was {test_set_accuracy:.2f}%")
```

```
The accuracy on the test set was 84.13%
```

Since our assignment is to use the train-validation-test schema, we will split once more the test data into a test and validation set

[8]:
```
Xtrain_t, Xval, ytrain_t, yval = train_test_split(Xtest, ytest, random_state =␣
 ↪random_state)
print(f"We have {Xtest.shape[0]} items in our test set")
print(f"We have {Xval.shape[0]} items in our validation set")
```

```
We have 334 items in our test set
We have 84 items in our validation set
```

Now we can save the depth of the tree with default hyperparameters. This way, we can vary the depths in order to see what is the best fit for our data.

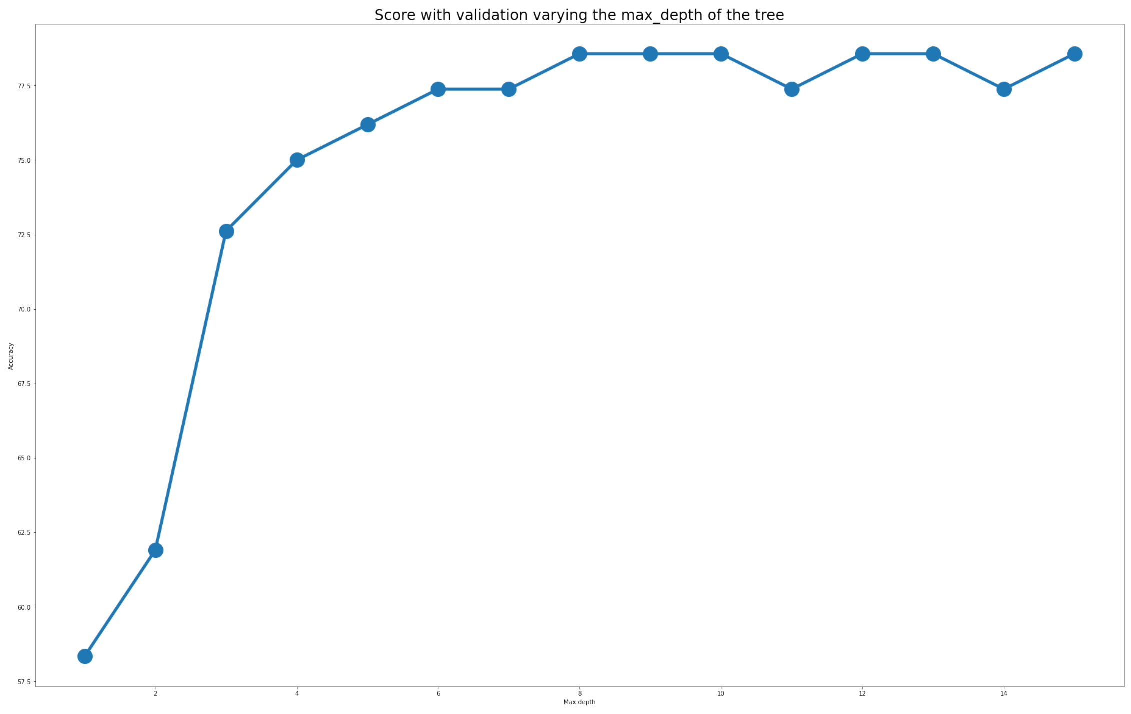
[9]:
```
default_tree_depth = estimator.tree_.max_depth
depths = range(1, default_tree_depth + 1)
```

[10]:
```
scores = []
for depth in depths:

    # Create a Decision Tree with limited depth
    estimator = tree.DecisionTreeClassifier(criterion="entropy", max_depth =␣
 ↪depth, random_state = random_state)

    # Fit it on our test data
    estimator.fit(Xtrain_t, ytrain_t)

    # Try predicting the class and save the accuracy of this classifier
    prediction = estimator.predict(Xval)
    score =  accuracy_score(yval, prediction) * 100
    scores.append(score)
```

1.1.3 3. The optimal parameter(s) (5pt)

We now have a look at the accuracy scores that we obtained in the previous step

[11]:
```
plt.figure(figsize=(32,20))
plt.plot(depths, scores, '-o', linewidth=5, markersize=24)
plt.xlabel('Max depth')
plt.ylabel('Accuracy')
plt.title("Score with validation varying the max_depth of the tree", fontsize =␣
 ↪24)
plt.show();
```



The best hyperparameter configuration is the one that maximises the accuracy

```
[12]: best_depth = depths[np.argmax(scores)]

      # We must remove 1 from best depth in order to obtain the correct score, as its␣
       ↪"zero" value is 1
      print(f"The best depth parameter was {best_depth}, with an accuracy of␣
       ↪{scores[best_depth - 1]:.2f}%")
```

```
The best depth parameter was 8, with an accuracy of 78.57%
```

We now test our tuned hyperparameter on the old training data and compute its accuracy

```
[13]: estimator = tree.DecisionTreeClassifier(criterion="entropy", max_depth =␣
       ↪best_depth, random_state = random_state)
      estimator.fit(Xtrain, ytrain)
      prediction = estimator.predict(Xtest)
      print(f"The accuracy of the tuned Decision Tree is {accuracy_score(ytest,␣
       ↪prediction) * 100:.2f}%")
```

```
The accuracy of the tuned Decision Tree is 84.13%
```

The results show that we obtained the same accuracy but with a smaller tree

```
[14]: print(f"The depth of the original tree was {default_tree_depth} and it is now␣
       ↪{best_depth}")
```

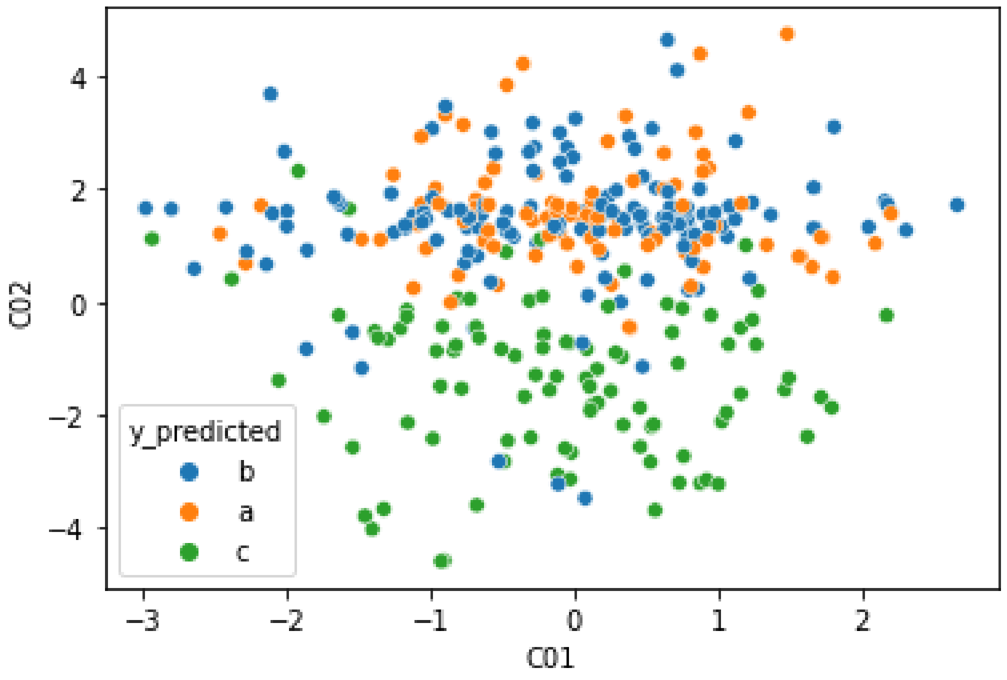
```
The depth of the original tree was 15 and it is now 8
```

1.1.4 4. A scatter plot of the test set using a pair of attributes of your choice with the class as colour (5pt)

Let us choose `C01` and `C02` as the attribute pair we will consider

[15]:
```python
attributes = ['C01', 'C02']

# We want to plot the test set, so we need to add the target column
# to the dataframe in order to use it as hue
df_test = Xtest.assign(y_predicted = prediction)

sns.scatterplot(x = attributes[0], y = attributes[1], data = df_test, hue =␣
 ↪'y_predicted')
```

[15]:
```
<AxesSubplot:xlabel='C01', ylabel='C02'>
```

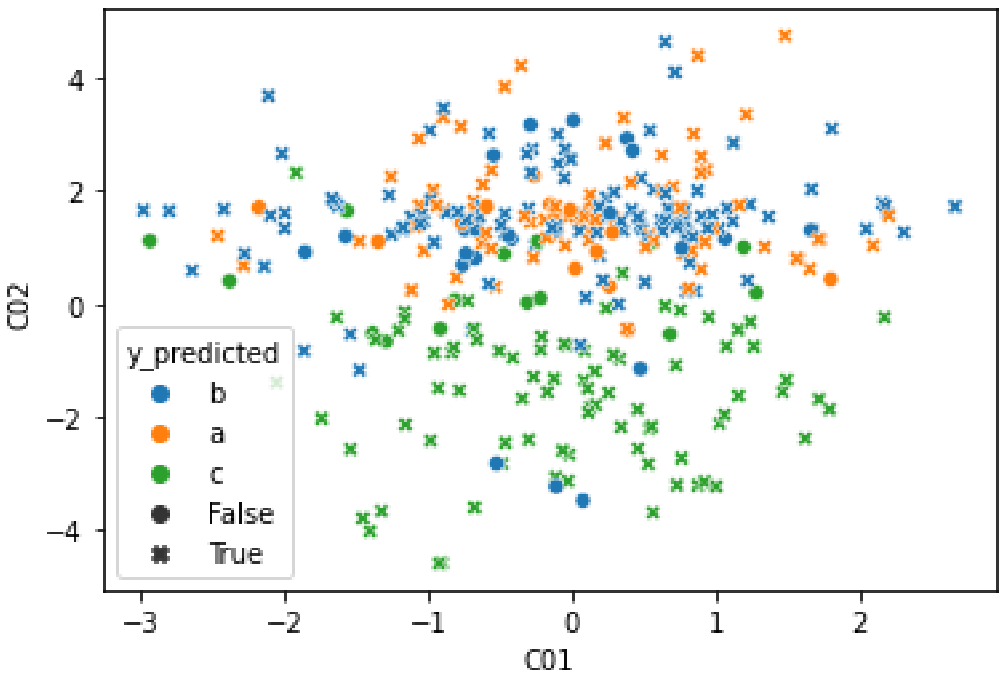


We notice that the items of class `c` tend to be in the lower part of the plot

1.1.5 5 ... and the good/bad prediction as the point style (5pt)

[16]:
```python
sns.scatterplot(x = attributes[0], y = attributes[1], data = df_test, hue =␣
 ↪'y_predicted', style = ytest == df_test['y_predicted'])
```

[16]:
```
<AxesSubplot:xlabel='C01', ylabel='C02'>
```


y_predicted
b
a
c
False
True
C01
C02
4
2
0
−2
−4
−3
−2
−1
0
1
2